AN INTRODUCTION TO SUM-PRODUCT NETWORKS

Renato Lui Geh
Computer Science
Institute of Mathematics and Statistics
University of São Paulo
renatolg@ime.usp.br

Abstract. Sum-Product Networks (SPNs) are deep probabilistic graphical models (PGMs) that compactly represent tractable probability distributions. Exact inference in SPNs is computed in time linear in the number of edges, an attractive feature that distinguishes SPNs from other PGMs. However, learning SPNs is a tough task. There have been many advances in learning both the structure and parameters of SPNs in the past few years. One interesting feature is the fact that we can make use of SPN's deep architecture and perform deep learning on these models. Since the number of hidden layers not only does not negatively impact the tractability of inference of SPNs but also augments the representability of this model, it is very much desirable to continue research on deep learning of SPNs. In this article we seek to produce a tutorial on Sum-Product Networks in a simpler, clearer way then how it is currently written in literature. We will introduce SPNs and explain how knowledge is represented in this model, how to perform exact inference and describe and analyse in detail a simple structural learning algorithm.

Keywords cluster analysis; data mining; probabilistic graphical models; tractable models; machine learning; deep learning

1. Introduction

Conventional probabilistic graphical models (PGMs) can compactly represent complex probability distributions and perform sub-exponential time inference through approximate methods. They are able to learn from data accurately and have very expressive semantics. However, exact inference in the general case is intractable. The alternative to exact inference is through the use of approximation algorithms. Unfortunately, approximate inference is at times unpredictable and analysis of these algorithms is very difficult.

Sum-Product Networks (SPNs) are deep PGMs that are able to compactly represent tractable probability distributions. Inference in SPNs is computed in time linear in the number of edges of the graph, where the number of edges is at most polynomial in the number of variables of the distribution. SPNs are represented by a DAG where internal nodes are either sum or product nodes. Leaf nodes are univariate distributions, though recent work on SPNs have shown that multivariate distributions are also allowed as leaves [RL14]. Learning of SPNs can be achieved

through subsequent clusterings of both variables and instanciations, where sum nodes can be seen as mixtures of distributions and product nodes as variable independencies. An interesting feature of SPNs is its deep architecture. As shown in Delalleau and Bengio's work [DB11], deep SPNs have more representative power than shallow SPNs. Poon and Domingos, on the innaugural SPN article [PD11], were able to learn accurate deep SPNs with 36 layers, as opposed to the few, less than ten layers that is typically learned in other deep models.

In this article we will provide a comprehensive description of discrete Sum-Product Networks, from its graph representation and how to perform exact polynomial time inference, to describing and analysing our implementation of the structural learning algorithm introduced in [GD13], a learning algorithm that is able to learn SPNs of potentially tens of layers.

2. Sum-Product Networks

A Sum-Product Network is a DAG where nodes can either be sums, products or leaves. In this section we will present two definitions of SPNs. The first one can be considered "low level", whilst the second carries heavier semantics due to properties we shall enumerate in this section. These classifications of "low level" vs "high level" will become clearer as we progress through this section.

2.1. *Network Polynomials*

Before we define SPNs more formally, we need to understand network polynomials. First introduced by Darwiche [Dar03], a network polynomial is a multilinear function over the variables of an unnormalized distribution. Computing the marginals of the distribution through its network polynomial is possible by setting all indicator variables to values consistent with evidence. The partial derivatives of the network polynomial can be seen as conditioning the network polynomial to a given event.

Network polynomials are not that attractive by themselves, as the number of terms in the polynomial is exponential in the number of variables. Arithmetic circuits (ACs) are a way of representing network polynomials with a polynomial number of terms. However, ACs are just another way of compiling a Bayesian network into a polynomial-sized model for tractable distributions.

Definition 2.1 (Network polynomial)**.** *Let $\mathcal{N}$ be a Bayesian network over set of variables $\mathbf{X}$. Let $Pa(X)$ be the set of parents of variable X. The network polynomial of $\mathcal{N}$ is given by*

$$f = \sum_{\mathbf{x}} \prod_{X, Pa(X) \sim \mathbf{x}} \lambda_X \theta_{X|Pa(X)}$$

where $\mathbf{x}$ is the set of possible instantiations of $\mathbf{X}$ and $X \sim x$ means that X is consistent with instantiation x.

Example 2.1. *Let $\mathcal{N} = A \to B$ be a Bayesian network where $\mathbf{X} = \{A, B\}$ is the set of binary variables of $\mathcal{N}$. The network polynomial of $\mathcal{N}$ is given by the*

multilinear function

$$f = \lambda_a\lambda_b\theta_{b|a}\theta_a + \lambda_a\lambda_{\overline{b}}\theta_{\overline{b}|a}\theta_a + \lambda_{\overline{a}}\lambda_b\theta_{b|\overline{a}}\theta_{\overline{a}} + \lambda_{\overline{a}}\lambda_{\overline{b}}\lambda_{\overline{b}|\overline{a}}\lambda_{\overline{a}}$$

The indicator variables of a variable X represent the possible configurations of X. Let $\mathrm{Val}(X)$ be the set of possible values of variable X and $m = |\mathrm{Val}(X)|$. Then the set of indicator variables of X, denoted by $\boldsymbol{\lambda}_X$ has elements $\lambda_X^1, \lambda_X^2, \ldots, \lambda_X^m$. A variable X is consistent with instantiation x if the indicator variable that agrees with x is set to 1 and the other indicator variables of X are set to 0. If there is no instantiation concerning variable X, then all indicator variables of X are set to 1.

Example 2.2. *Let* $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$ *be the set of binary variables of a network polynomial* f*. A variable* X_i *can be evaluated as either* 0 *or* 1*. Let* $\mathbf{e} = \{X_1 = 1, X_2 = 0, X_3 = 1\}$ *be the evidence set. Computing the marginals of* f *can be done by setting all indicator variables consistent with* $\mathbf{e}$

$$\begin{aligned} \lambda_{X_1}^0 &= 0 & \lambda_{X_1}^1 &= 1 \\ \lambda_{X_2}^0 &= 1 & \lambda_{X_2}^1 &= 0 \\ \lambda_{X_3}^0 &= 0 & \lambda_{X_3}^1 &= 1 \\ \lambda_{X_4}^0 &= 1 & \lambda_{X_4}^1 &= 1 \end{aligned}$$

The partial derivative of a network polynomial by a certain variable corresponds to conditioning the polynomial to a certain event. By computing $\partial f/\partial\lambda_X(\mathbf{e})$, we are merely setting the indicator variables consistently with $\mathbf{e}$. This leads to the theorem:

Theorem 2.1. *Let* $\mathcal{N}$ *be a Bayesian network representing probability distribution* Pr *and having network polynomial* f*. For every variable* X *and evidence* $\mathbf{e}$*, we have*

$$\frac{\partial f}{\partial \lambda_X}(\mathbf{e}) = \Pr(x, \mathbf{e} \setminus \{X\})$$

2.2. *Representing Sum-Product Networks*

Sum-Product Networks are closely related to network polynomials and arithmetic circuits. It can be seen as a way of representing the network polynomial. Graphically, a Sum-Product Network is a DAG with internal nodes as sums and products and leaves as indicator variables.

Definition 2.2. *An SPN* S *is a DAG with two types of internal nodes: sums and products. A sum node is a node whose value is defined by* $v_i = \sum_{j \in Ch(i)} w_{ij} v_j$*, where* v_i *is the value of sum node* i*,* $Ch(i)$ *is the set of children of node* i*,* w_{ij} *is the non-negative weight associated with directed edge* $i \to j$ *and* v_j *is the value of node* j*. The value of a product node* i *is given by* $v_i = \prod_{j \in Ch(i)} v_j$*. A leaf node is always an indicator variable. Its value is the value of the indicator variable. The value of* S *is the value of the root node of* S*.*

An arbitrary node i of an SPN S is itself an SPN. We will denote the sub-SPN rooted at i as S_i. Although the weights of a sum node may take any non-negative

value, if all the weights from a sum node sum to one, then that node is normalized. If all sum nodes in the SPN are normalized, then the SPN is normalized and its partition function will be 1. Otherwise the partition function is the value of S when all indicator variables are set to 1. Such function will be denoted as $S(*)$. It was recently proven that normalized SPNs have as much representability power as unnormalized SPNs [Peh+15]. The scope of an SPN is the set of variables that are present in it. The same applies to each sub-SPN. A variable is in an SPN if any of its indicator variables is a leaf in that SPN. We will denote the scope of an SPN S as $\mathrm{Sc}(S)$. The elements of $\mathrm{Sc}(S)$ are the variables present in S.

Definition 2.3 (Validity)**.** *An SPN is valid if and only if it correctly computes the probability of evidence of the probability distribution it represents.*

Definition 2.4 (Completeness)**.** *An SPN S is complete if and only if, for every sum node i in S, all children of i have the same scope.*

Definition 2.5 (Consistency)**.** *An SPN S is consistent if and only if, for every product node i in S, a variable present in i has the same value as the same variable in the descendents of i.*

Theorem 2.2. *An SPN is valid if it is complete and consistent.*

This theorem states that an SPN computes its value correctly if it is complete and consistent. However, this does not exclude the possibility of correctly computing the probability of evidence of incomplete and inconsistent SPNs.

Definition 2.6 (Decomposability)**.** *An SPN S is decomposable if and only if, for every product node i in S, every pair of children of i have disjoint scopes with each other.*

Corollary 2.1. *Decomposability implies consistency.*

Proof. It is easy to see that decomposability implies consistency. For each product node i in S, let u and v be a pair of children of i. If S is decomposable, then $\mathrm{Sc}(u) \cap \mathrm{Sc}(v) = \emptyset$ and therefore no variable has a contradicting value, since no child of i shares a variable in common. □

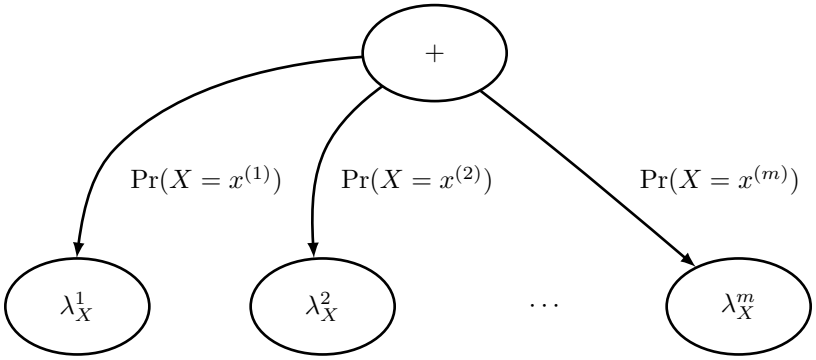
Because of Proposition 2.1, we can create complete and decomposable SPNs and they will always be valid. We rather work with decomposability instead of consistency because decomposable SPNs are easier to create than solely consistent ones. Peharz et al proved that all complete and consistent SPNs can be transformed into a complete and decomposable SPN with no more than a polynomially large addition of nodes and edges [Peh+15]. Furthermore, a valid yet non-decomposable SPN may not compute a network polynomial in some cases. Intuitively, decomposable SPNs can be seen as independency between sets of variables, which has a much stronger semantic than consistency.

2.3. *Sum-Product Networks as Combinations of Univariate Distributions*

Let Pr be a normalized univariate distribution over variable X, $\mathrm{Val}(X)$ be the possible instantiations of X and $m = |\mathrm{Val}(X)|$. The network polynomial of Pr is given by

$$f = \lambda_X^1 \Pr(X = x^{(1)}) + \lambda_X^2 \Pr(X = x^{(2)}) + \cdots + \lambda_X^m \Pr(X = x^{(m)})$$

The simplest SPN we can use to represent such network polynomial consists of a root sum node and its children as leaves, where each leaf is an indicator variable of X and each edge that comes from the sum node to a leaf i is the $\Pr(X = x^{(i)})$.



Note that this SPN is both complete, since the sum node's scope is $\{X\}$ and so are each of its children's scope; and decomposable, since there are no product nodes. We will call these univariate SPNs as uSPNs. With this in mind, we can now present another definition of SPNs.

Definition 2.7. *Recursively, an SPN can be either (mutually exclusive):*

(1) A univariate tractable probability distribution or;
(2) A weighted sum of SPNs where all weights are in the range $[0, 1]$ *and sum up to* 1*, and whose children all have same scope as the parent or;*
(3) A product of SPNs in which every child has disjoint scope with every other child.

Let $\mathbf{e}$ *be the set of evidence. The value of SPN (1) given* $\mathbf{e}$ *is the value of the underlying uSPN that represents (1) given* $\mathbf{e}$*. The value of SPN (2) is* $\sum_{j \in Ch(i)} w_{ij} S_j(\mathbf{e})$*. The value of SPN (3) is* $\prod_{j \in Ch(i)} S_j(\mathbf{e})$*.*

With this new definition we have abandoned indicator variables in favor of a "higher-level" view of SPNs, that is, of combinations of univariate distributions. Also notice that we implicitly define SPNs as complete, decomposable and normalized. Item (2) defines sum nodes as complete and normalized, and item (3) define product nodes as decomposable. This new definition also represent a more intuitive view of SPNs. It is clear from this definition that sum nodes act as instances

clustering, since all children have the same scope. In the same way, product nodes represent clustering of independent sets of variables, since children have disjoint scopes with one another.

3. Inference on Sum-Product Networks

References

[Dar03] Adnan Darwiche. "A Differential Approach to Inference in Bayesian Networks". In: (2003).

[DB11] Olivier Delalleau and Yoshua Bengio. "Shallow vs. Deep Sum-Product Networks". In: *Advances in Neural Information Processing Systems 24 (NIPS 1011)* (2011).

[GD13] Robert Gens and Pedro Domingos. "Learning the Structure of Sum-Product Networks". In: *International Conference on Machine Learning* 30 (2013).

[PD11] Hoifung Poon and Pedro Domingos. "Sum-Product Networks: A New Deep Architecture". In: *Uncertainty in Artificial Intelligence* 27 (2011).

[Peh+15] Robert Peharz et al. "On Theoretical Properties of Sum-Product Networks". In: *International Conference on Artificial Intelligence and Statistics 18 (AISTATS 2015)* (2015).

[RL14] Amirmohammad Rooshenas and Daniel Lowd. "Learning Sum-Product Networks with Direct and Indirect Variable Interactions". In: *International Conference on Machine Learning 31 (ICML 2014)* (2014).